**LAB REMIX**

**Exercise n.1**

Create, execute, deploy, and run your ERC 20 simple token with your

* Name- NameCoin ( es LauraCoin)
* Symbol- the initial of your name, and surname followed by X ( es NFX)
* Supply- on your choice ( es 1000, 5000)

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.5.17;

import "@0x/contracts-erc20/contracts/src/ERC20Token.sol";

/\*\*

\* @title SampleERC20

\* @dev Create a sample ERC20 standard token

\*/

contract SampleERC20 is ERC20Token {

string public name;

string public symbol;

uint256 public decimals;

uint256 public totalSupply;

constructor (

string memory \_name,

string memory \_symbol,

uint256 \_decimals,

uint256 \_totalSupply

)

public

{

name = \_name;

symbol = \_symbol;

decimals = \_decimals;

totalSupply = \_totalSupply;

balances[msg.sender] = \_totalSupply;

}

}

**Exercise n. 2 Voting**

Create, execute, deploy, and run your Voting App

pragma solidity >=0.4.22 <0.6.0;

contract Ballot {

struct Voter {

uint weight;

bool voted;

uint8 vote;

address delegate;

}

struct Proposal {

uint voteCount;

}

address chairperson;

mapping(address => Voter) voters;

Proposal[] proposals;

/// Create a new ballot with $(\_numProposals) different proposals.

constructor(uint8 \_numProposals) public {

chairperson = msg.sender;

voters[chairperson].weight = 1;

proposals.length = \_numProposals;

}

/// Give $(toVoter) the right to vote on this ballot.

/// May only be called by $(chairperson).

function giveRightToVote(address toVoter) public {

if (msg.sender != chairperson || voters[toVoter].voted) return;

voters[toVoter].weight = 1;

}

/// Delegate your vote to the voter $(to).

function delegate(address to) public {

Voter storage sender = voters[msg.sender]; // assigns reference

if (sender.voted) return;

while (voters[to].delegate != address(0) && voters[to].delegate != msg.sender)

to = voters[to].delegate;

if (to == msg.sender) return;

sender.voted = true;

sender.delegate = to;

Voter storage delegateTo = voters[to];

if (delegateTo.voted)

proposals[delegateTo.vote].voteCount += sender.weight;

else

delegateTo.weight += sender.weight;

}

/// Give a single vote to proposal $(toProposal).

function vote(uint8 toProposal) public {

Voter storage sender = voters[msg.sender];

if (sender.voted || toProposal >= proposals.length) return;

sender.voted = true;

sender.vote = toProposal;

proposals[toProposal].voteCount += sender.weight;

}

function winningProposal() public view returns (uint8 \_winningProposal) {

uint256 winningVoteCount = 0;

for (uint8 prop = 0; prop < proposals.length; prop++)

if (proposals[prop].voteCount > winningVoteCount) {

winningVoteCount = proposals[prop].voteCount;

\_winningProposal = prop;

}

}

}

**Exercise n.3 Hello Coin**

Create, execute, deploy!

pragma solidity ^0.4.18;

contract HelloCoin {

string public name = 'HelloCoin';

//currency name. Please feel free to change it

string public symbol = 'coin\_nadia';

//choose a currency symbol. Please feel free to change it

mapping (address => uint) balances;

//a key-value pair to store addresses and their account balances

event Transfer(address \_from, address \_to, uint256 \_value);

// declaration of an event. Event will not do anything but add a record to the log

constructor() public {

//when the contract is created, the constructor will be called automatically

balances[msg.sender] = 10000;

//set the balances of creator account to be 10000. Please feel free to change it to any //number you want.

}

function sendCoin(address \_receiver, uint \_amount) public returns(bool sufficient) {

if (balances[msg.sender] < \_amount) return false;

// validate transfer

balances[msg.sender] -= \_amount;

balances[\_receiver] += \_amount;

emit Transfer(msg.sender, \_receiver, \_amount);

// complete coin transfer an

return true;

}

function getBalance(address \_addr) public view returns(uint) {

//balance check

return balances[\_addr];

}

}

**Exercise n.4**

In Remix, modify the code below so that the sender function will be able to transfer tokens to three accounts at the same time instead of one, splitting the amount into three. Please implement the required changes, compile, deploy, and test it with Remix. Report a screenshot of the new function. Explain also your modification to the code.

pragma solidity ^0.5.0;

contract Coin {

// The keyword "public" makes those variables

// easily readable from the outside.

address public minter;

mapping (address => uint) public balances;

// Events allow light clients to react to

// changes efficiently.

event Sent(address from, address to, uint amount);

// This is the constructor whose code is

// run only when the contract is created.

constructor() public {

minter = msg.sender;

}

function mint(address receiver, uint amount) public {

require(msg.sender == minter);

require(amount < 1e60);

balances[receiver] += amount;

}

function send(address receiver, uint amount) public {

require(amount <= balances[msg.sender], "Insufficient balance.");

balances[msg.sender] -= amount;

balances[receiver] += amount;

emit Sent(msg.sender, receiver, amount);

}

}

Immagine che contiene testo, elettronica, schermata, software

Il contenuto generato dall'IA potrebbe non essere corretto.

**Exercise n.5**

**Script for creating NFT 721 with OpenZeppelin**

**// SPDX-License-Identifier: MIT**

**pragma solidity ^0.8.9;**

**import "@openzeppelin/contracts/token/ERC721/ERC721.sol";**

**contract MyToken is ERC721 {**

**constructor() ERC721("MyToken", "MTK") {}**

**}**

**Exercise n. 6 DIGITAL VENDING MACHINES**

pragma solidity 0.8.7;

contract VendingMachine {

// Declare state variables of the contract

address public owner;

mapping (address => uint) public cupcakeBalances;

// When 'VendingMachine' contract is deployed:

// 1. set the deploying address as the owner of the contract

// 2. set the deployed smart contract's cupcake balance to 100

constructor() {

owner = msg.sender;

cupcakeBalances[address(this)] = 100;

}

// Allow the owner to increase the smart contract's cupcake balance

function refill(uint amount) public {

require(msg.sender == owner, "Only the owner can refill.");

cupcakeBalances[address(this)] += amount;

}

// Allow anyone to purchase cupcakes

function purchase(uint amount) public payable {

require(msg.value >= amount \* 1 ether, "You must pay at least 1 ETH per cupcake");

26

require(cupcakeBalances[address(this)] >= amount, "Not enough cupcakes in stock to complete this purchase");

27

cupcakeBalances[address(this)] -= amount;

28

cupcakeBalances[msg.sender] += amount;

29

}

30

}

Show all

📋 Copy

Like how a vending machine removes the need for a vendor employee, smart contracts can replace interm